PART B

MORE ABOUT FUNCTIONS

JAVASCRIPT

2023 © Braintio ltd | Confidential

Callback

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

```
<html>
   <body>
       <script>
               function greeting(name) {
                   alert('Hello ' + name);
               }

               function processUserInput(callback) {
                   var name = prompt('Please enter your name.');
                   callback(name);
               }
               processUserInput(greeting);
       </script>
   </body>
</html>
```

Callback

```
<html>
   <body>
       <script>
           function add(n1, n2){
           console.log(n1+n2);
           }
           function sub(n1, n2){
           console.log(n1-n2);
           }
           function mult(n1, n2){
           console.log(n1*n2);
           }
           function f(n1, n2, callback, callback2){
           callback(n1,n2);
           callback2(n1,3);
           }
           f(5,6,mult,add);
       </script>
   </body>
</html>
```

Function return function

```
<html>
    <body>
        <script>
            function magic() {
                return function calc(x) { return x * 42; };
            }

            var answer = magic();
            answer(1337); // 56154
        </script>
    </body>
</html>
```

Call, apply and bind

Call, apply and bind functions are all used to change the scope of what `this` is equal to inside of a function or a method.

Call

```
<html>
   <body>
       <script>
 let  course= {
 name: '',
 description: '',
 students:[],
 addStudents(studentName){
       this.students.push(studentName)
       console.log(`${studentName} added to ${this.name}
course`)
 },
 date: '12/12/2021'
};
let english = {
   name:"english course",
   description:"this is good course",
   students:[]
}
let math = {
   name:"math course",
   description:"this is very good course",
   students:[]
}
let addStudents = math.addStudents;
//add("asaf")//its will not work.
//this function will referance to
undefined
addStudents.call(english,"asaf")
addStudents.call(math,"Dani")
addStudents.call(english,"asaf")
addStudents.call(math,"Ron")

console.log(math);
console.log(english);

       </script>
   </body>
</html>
```

Call

The call function creates a new instance of the course object called english and then the this inside the `addStudents` function will point to the new object.

```
addStudents(studentName){
        this.students.push(studentName)
        console.log(`${studentName} added to
${this.name} course`)
```

```
let english = {
   name:"english course",
   description:"this is good
course",
   students:[]
}
```

```
let math = {
   name:"math course",
   description:"this is very good
course",
   students:[]
}
```

```
let addStudents = math.addStudents;
//addStudents("asaf")//its will
not work.
//this function will referance to
undefined
addStudents.call(english,"asaf")
addStudents.call(math,"Dani")
addStudents.call(english,"asaf")
addStudents.call(math,"Ron")

console.log(math);
console.log(english);

        </script>
    </body>
</html>
```

Apply is the same but the function get an array parameter

```
<html>
   <body>
       <script>
 let  course= {
 name: '',
 description: '',
 students:[],
 addStudents(studentName){
       this.students.push(studentName)
       console.log(`${studentName} added to ${this.name}
course`)
 },
 date: '12/12/2021'
};
let english = {
   name:"english course",
   description:"this is good course",
   students:[]
}
let math = {
   name:"math course",
   description:"this is very good course",
   students:[]
}
let addStudents = math.addStudents;
//add("asaf")//its will not work.
//this function will referance to
undefined
addStudents.apply(english,["asaf"])
addStudents.apply(math,["Dani"])
addStudents.apply(english,["asaf"])
addStudents.apply(math,["Ron"])

console.log(math);
console.log(english);
studentData=['menny']
addStudents.call(english,...studentData)
console.log(english);

       </script>
   </body>
</html>
```

Apply is the same but the function get an array parameter

The apply function creates a new instance of the course object called english and then the this within the `addStudents` function will point to the new object.

```
addStudents(studentName){
        this.students.push(studentName)
        console.log(`${studentName} added to
${this.name} course`)
```

```
let english = {
   name:"english course",
   description:"this is good
course",
   students:[]
}
```

```
let math = {
   name:"math course",
   description:"this is very good
course",
   students:[]
}
```

```
let addStudents = math.addStudents;
//add("asaf")//its will not work.
//this function will referance to
undefined
addStudents.apply(english,["asaf"])
addStudents.apply(math,["Dani"])
addStudents.apply(english,["asaf"])
addStudents.apply(math,["Ron"])

console.log(math);
console.log(english);
studentData=['menny']
addStudents.call(english,...studentData)
console.log(english);
```

array

```
        </script>
    </body>
</html>
```

bind

```html
<html>
   <body>
       <script>
 let  course= {
 name: '',
 description: '',
 students:[],
 addStudents(studentName){
       this.students.push(studentName)
       console.log(`${studentName} added to ${this.name}
course`)
 },
 date: '12/12/2021'
};
let english = {
   name:"english course",
   description:"this is good course",
   students:[]
}
let math = {
   name:"math course",
   description:"this is very good course",
   students:[]
}
let addStudents = math.addStudents;
//add("asaf")//its will not work.
//this function will referance to
undefined
            //bind
let addToEnglsihStudents =
addStudnents.bind(english);
addToEnglsihStudents("Lara")
addToEnglsihStudents("Lisa")
addToEnglsihStudents("Morgan")

       </script>
   </body>
</html>
```

bind

הפונקציה apply יוצרת מופע חדש של האוביקט course בשם english ואז ה this
שבתוך פונקצית **addStudents** יצביע לאובקיט החדש.

```
addStudents(studentName){
        this.students.push(studentName)
        console.log(`${studentName} added to
${this.name} course`)
```

```
let english = {
   name:"english course",
   description:"this is good
course",
   students:[]
}
```

```
let math = {
   name:"math course",
   description:"this is very good
course",
   students:[]
}
```

1. Create new object english or math
2. Now you can pass parameter to the function

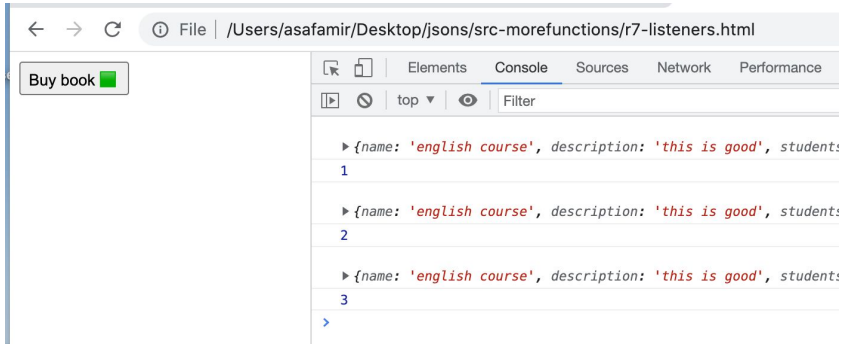
```
let addStudents = math.addStudents;
//add("asaf")//its will not work.
//this function will referance to
undefined
            //bind

let addToEnglsihStudents =
addStudnents.bind(english);
addToEnglsihStudents("Lara")
addToEnglsihStudents("Lisa")
addToEnglsihStudents("Morgan")


        </script>
    </body>
</html>
```

Bind and dom

```
<html>
   <body>
       <button class="buy">Buy book 🟩</button>
       <script>
 let  course= {
 name: '',
 books: 0,
 description: '',
 students:[],
 addStudents(studentName){
       this.students.push(studentName)
       console.log(`${studentName} added to ${this.name} course`)
 },
 date: '12/12/2021'
};
let english = {
   name:"english course",
   books: 0,
   description:"this is good course",
   students:[]
}
english.buyCourse = function(){
 this.books++
 console.log(this.books)// 1,2,3,,,,n
}
//document.querySelector('.buy').addEventListener('click', english.buyCourse)
document.querySelector('.buy').addEventListener('click',english.buyCourse.bind(english))
```



Add function to the english object

You have to bind the new function to the instance of the object

CLOSURE

JAVASCRIPT

2023 © Braintio ltd | Confidential

Closure

closure היא שילוב של פונקציה bundled יחד (סגורה) עם הפניות למצב הסובב אותה (הסביבה המילונית).
במילים אחרות, closure מעניקה לך גישה להיקף של פונקציה חיצונית מתוך פונקציה פנימית.
ב-JavaScript, ה closure נוצרות בכל פעם שפונקציה נוצרת, בזמן יצירת הפונקציה.

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives you access to an outer function’s scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

Closure

```
function init() {
  var name = 'Mozilla'; // name is a local variable created by init
  function displayName() { // displayName() is the inner function, a closure
    alert(name); // use variable declared in the parent function
  }
  displayName();
}
init();
```

- init() creates a local variable called name and a function called
- displayName(). The displayName() function is an inner function that is defined inside init() and is available only within the body of the init() function.
- Note that the displayName() function has no local variables of its own. However, since inner functions have access to the variables of outer functions, displayName() can access the variable name declared in the parent function, init().

2023 © Braintio ltd | Confidential

Closure

Running this code has exactly the same effect as the previous example of the init() function above. What's different (and interesting) is that the displayName() inner function is returned from the outer function before being executed.

```
<html>
   <body>
       <script>
           function makeFunc() {
               var name = 'Oren';
               function displayName() {
                   alert(name);
               }
               return displayName;
           }
           var myFunc = makeFunc();
           myFunc();
       </script>
   </body>
</html>
```

Closure

```
<html>
   <body>
       <script>
           function adding(x) {
               return function sum(y){
               return x + y;
           }
           }
           var add10 = adding(10);
           var add15 = adding(15);
           console.log(add10(2)); //??
           console.log(add15(2)); //???
       </script>
   </body>
</html>
```

Closure

```html
<html>
    <body>
        <script>
            function adding(x) {
                return function sum(y){
                return x + y;
            }
            }
            var add10 = adding(10);
            var add15 = adding(15);
            console.log(add10(2)); //12
            console.log(add15(2)); //17
        </script>
    </body>
</html>
```

In this example, we have defined a function `adding(x)`, that takes a single argument `x`, and returns a new function. The function it returns takes a single argument `y`, and returns the sum of `x` and `y`.

In essence, **`adding`** is a function factory. It creates functions that can add a specific value to their argument. In the above example, the function factory creates two new functions—one that adds five to its argument, and one that adds 10.

`add5` and `add10` are both closures. They share the same function body definition, but store different lexical environments. In `add5`'s lexical environment, `x` is 5, while in the lexical environment for `add10`, `x` is 10.

Practical Closure

```
body {
  font-family: Helvetica, Arial, sans-serif;
  font-size: 12px;
}

h1 {
  font-size: 1.5em;
}

h2 {
  font-size: 1.2em;
}

function makeSizer(size) {
  return function() {
    document.body.style.fontSize = size +
'px';
  };
}

var size12 = makeSizer(12);
var size14 = makeSizer(14);
var size16 = makeSizer(16);
```

Closures are useful because they let you associate data with a function that operates on that data.

We use a closure anywhere that you might normally use an object with only a single method.

While programming web related programs - You define some behaviour, and then attach it to an event that is triggered by the user (such as a click or a keypress). The code is attached as a callback (a single function that is executed in response to the event).

For example, if we want to add buttons to a page to adjust the text size, we specify the font-size of the body element (in pixels), and then set the size of the other elements on the page (such as headers) using the relative em unit:

Practical Closure

```
body {
  font-family: Helvetica, Arial, sans-serif;
  font-size: 12px;
}

h1 {
  font-size: 1.5em;
}

h2 {
  font-size: 1.2em;
}

function makeSizer(size) {
  return function() {
    document.body.style.fontSize = size +
'px';
  };
}

var size12 = makeSizer(12);
var size14 = makeSizer(14);
var size16 = makeSizer(16);
```

size12, size14, and size16 are now functions that resize the body text to 12, 14, and 16 pixels, respectively. You can attach them to buttons (in this case hyperlinks) as demonstrated in the following code example.

```
document.getElementById('size-12').onclick = size12;

document.getElementById('size-14').onclick = size14;

document.getElementById('size-16').onclick = size16;
```

Or

```
<a href="#" id="size-12">12</a>

<a href="#" id="size-14">14</a>

<a href="#" id="size-16">16</a>
```

Closure for Advance good to know

```
<html>
    <body>
        <script>
        var counter = (function() {
        var privateCounter = 0;
        function changeBy(val)
        {
            privateCounter += val;
        }
        return {
        increment: function() {
        changeBy(1);
        },
        decrement: function() {
        changeBy(-1);
        },
        value: function() {
        return privateCounter;
        }
        }
    })();
```

```
    console.log(counter.value()); // logs 0
    counter.increment();
    counter.increment();
    console.log(counter.value()); // logs 2
    counter.decrement();
    console.log(counter.value()); // logs 1
        </script>
    </body>
</html>
```

Array And Map For Each

JAVASCRIPT

2023 © Braintio ltd | Confidential

סריקה על ידי forEach

```
<html>
    <body>
        <script>
        const array1 = ['a', 'b', 'c'];
        array1.forEach(element =>
console.log(element));
        // expected output: "a"
        // expected output: "b"
        // expected output: "c"
        </script>
    </body>
</html>
```

- השיטה forEach() מבצעת פונקציה מסופקת(privided) פעם אחת עבור כל רכיב מערך.

סריקה על ידי foreach

```html
<html>
   <body>
       <script>
           const items = ['item1', 'item2', 'item3']
           const copyItems = []
           // for
           for (let i = 0; i < items.length; i++) {
           copyItems.push(items[i])
           }

       </script>
   </body>
</html>
```

```html
<html>
   <body>
       <script>
          const items = ['item1', 'item2', 'item3']
           //forEach
           items.forEach(function(item){
           copyItems.push(item)
            })
       </script>
   </body>
</html>
```

סריקה על ידי foreach

```
<html>
   <body>
       <script>
           // Notice that index 2 is skipped, since there is no item
at
           // that position in the array...
           [2, 5, , 9].forEach( function(element, index, array) {
           console.log('a[' + index + '] = ' + element)
           })
           // logs:
           // a[0] = 2
           // a[1] = 5
           // a[3] = 9

       </script>
   </body>
</html>
```

callbackFn is invoked with three arguments:

1. the value of the element
2. the index of the element
3. the Array object being traversed

סריקה על ידי foreach

```
<html>
    <body>
        <script>
            let words = ['one', 'two', 'three', 'four']
            words.forEach(function(word) {
            console.log(word)
            if (word === 'two') {
                words.shift() //'one' will delete from array
            }
            }) // one // two // four
            console.log(words);  //['two', 'three', 'four']
        </script>
    </body>
</html>
```

Looping map forEach

שיטת Map.forEach משמשת ללולאה על המפה עם הפונקציה הנתונה ומבצעת את הפונקציה הנתונה על פני כל זוג מפתח-ערך.

Syntax:

```
myMap.forEach(callback, value, key, thisArg)
```

Parameters: This method accepts four parameters as mentioned above and described below:

- **callback:** This is the function that executes on each function call.
- **value:** This is the value for each iteration.
- **key:** This is the key to reach iteration.
- **thisArg:** This is the value to use as this when executing callback.

```
<script>
    // Creating a map using Map object
    let mp=new Map()

    // Adding values to the map
    mp.set("a",1);
    mp.set("b",2);
    mp.set("c",3);

    // Logging map object to console
    mp.forEach((values,keys)=>{
      document.write(values,keys+"<br>")
    })
</script>
```

Output:

```
1a
2b
3c
```

Looping map forEach

The forEach method executes the provided callback once for each key of the map which actually exist. It is not invoked for keys which have been deleted. However, it is executed for values which are present but have the value undefined.

```
<html>
   <body>
       <script>
       function logMapElements(value, key, map) {
       console.log(`m[${key}] = ${value}`);
       }

       new Map([['foo', 3], ['bar', {}], ['baz', undefined]])
       .forEach(logMapElements);

// expected output: "m[foo] = 3"
// expected output: "m[bar] = [object Object]"
// expected output: "m[baz] = undefined"
       </script>
   </body>
</html>
```

callback is invoked with three arguments:

- the entry's value
- the entry's key
- the Map object being traversed

Looping set forEach

The forEach() method executes a provided function once for each value in the Set object, in insertion order.

The forEach() method executes the provided callback once for each value which actually exists in the Set object. It is not invoked for values which have been deleted. However, it is executed for values which are present but have the value undefined.

callback is invoked with three arguments:

- the element value
- the element key
- the Set object being traversed

There are no keys in Set objects, however, so the first two arguments are both values contained in the Set. This is to make it consistent with other forEach() methods for Map and Array.

Looping set forEach

```
<html>
   <body>
       <script>
           function logSetElements(value1, value2, set) {
           console.log(`s[${value1}] = ${value2}`);
           }

           new Set(['foo', 'bar', undefined]).forEach(logSetElements);

           // expected output: "s[foo] = foo"
           // expected output: "s[bar] = bar"
           // expected output: "s[undefined] = undefined"
       </script>
   </body>
</html>
```

callback is invoked with three arguments:

- the element value
- the element key
- the Set object being traversed

תרגול Looping array forEach

כתוב קוד שמדפיס את הפלט למטה:

```
1 x 1 = 1
2 x 2 = 4
3 x 3 = 9
4 x 4 = 16
5 x 5 = 25
```

הוראות

1. צור מערך ספציפי
2. השתמש בו עם forEach()
3. בונוס: הדפס את המערך כדי לראות אם המערך השתנה

פתרון: Looping array forEach practice

JS:

```
//an array of numbers
let numberArray = [1, 2, 3, 4, 5];

//output the square of each number
let returnValue = numberArray.forEach(num =>
  console.log(`${num} x ${num} = ${num * num}`)
);

//the array hasn't changed
console.log(numberArray);
```

Output:

```
1 x 1 = 1
2 x 2 = 4
3 x 3 = 9
4 x 4 = 16
5 x 5 = 25
[1,2,3,4,5]
```

PART B

Map, **Filter**, Reduce **And** Find

JAVASCRIPT

2023 © Braintio ltd | Confidential

Map

שיטת map() יוצרת מערך חדש המאוכלס בתוצאות של קריאה לפונקציה שסופקה בכל אלמנט במערך הקורא.

```
<html>
   <body>
       <script>
           const array1 = [1, 4, 9, 16];
           // pass a function to map
           const map1 = array1.map(x => x * 2);
           console.log(map1);
           // expected output: Array [2, 8, 18, 32]
       </script>
   </body>
</html>
```

Map

map קורא לפונקציית callbackFn שסופקה פעם אחת עבור כל אלמנט במערך, לפי הסדר, ובונה מערך חדש מהתוצאות. callbackFn מופעל רק עבור אינדקסים של המערך אשר הוקצו להם ערכים (כולל לא מוגדר).

הקוד הבא לוקח מערך של מספרים ויוצר מערך חדש המכיל את השורשים הריבועיים של המספרים במערך הראשון.

```
<html>
   <body>
       <script>
          // Mapping an array of numbers to an array of square roots
          let numbers = [1, 4, 9]
          let roots = numbers.map(function(num) {
               return Math.sqrt(num)
           })
           // roots is now     [1, 2, 3]
           // numbers is still [1, 4, 9]
       </script>
   </body>
</html>
```

Map

מיפוי מערך של מספרים באמצעות פונקציה המכילה ארגומנט.

הקוד הבא מראה כיצד המפה פועלת כאשר משתמשים איתה בפונקציה הדורשת ארגומנט אחד. הארגומנט יוקצה אוטומטית מכל אלמנט של המערך כלולאות(array as map loops) מפה דרך המערך המקורי.

```
<html>
   <body>
       <script>
           let numbers = [1, 4, 9]
           let doubles = numbers.map(function(num)  {
           return num * 2
           })

           // doubles is now   [2, 8, 18]
           // numbers is still [1, 4, 9]
       </script>
   </body>
</html>
```

Filter

שיטת filter() יוצרת מערך חדש עם כל האלמנטים שעוברים את הבדיקה המיושמת על ידי הפונקציה שסופקה.

```
<html>
   <body>
       <script>
           const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction',
'present'];

           const result = words.filter(word => word.length > 6);

           console.log(result);
           // expected output: Array ["exuberant", "destruction", "present"]
       </script>
   </body>
</html>
```

Filter

filter() קורא לפונקציית callbackFn שסופקה פעם אחת עבור כל אלמנט במערך, ובונה מערך חדש של כל הערכים שעבורם callbackFn מחזיר ערך שכופה ל-true. callbackFn מופעל רק עבור אינדקסים של המערך שהוקצו להם ערכים; זה לא מופעל עבור אינדקסים שנמחקו או שמעולם לא הוקצו להם ערכים. רכיבי מערך שאינם עוברים את מבחן callbackFn מדלגים, ואינם נכללים במערך החדש.

callbackFn is invoked with three arguments:

1. the value of the element
2. the index of the element
3. the Array object being traversed

Filter

הדוגמה הבאה משתמשת ב-filter() כדי ליצור מערך מסונן שבו כל הרכיבים עם ערכים פחות מ-10 הוסרו.

```
<html>
   <body>
       <script>
           function isBigEnough(value) {
           return value >= 10
           }
           let filtered = [12, 5, 8, 130, 44].filter(isBigEnough)
           // filtered is [12, 130, 44]
       </script>
   </body>
</html>
```

Filter

הדוגמה הבאה משתמשת ב-filter() כדי ליצור מערך מסונן שבו כל הרכיבים עם ערכים פחות מ-10 הוסרו.

```
<html>
   <body>
       <script>
           function isBigEnough(value) {
           return value >= 10
           }
           let filtered = [12, 5, 8, 130, 44].filter(isBigEnough)
           // filtered is [12, 130, 44]
       </script>
   </body>
</html>
```

Filter

מצא את כל המספרים הראשוניים החיוביים הגדולים מ 0 במערך (ניתן להניח שהפונקציה מקבלת מספר גדול מ 0)
הדוגמה הבאה מחזירה את כל המספרים הראשוניים במערך:

```
<html>
   <body>
       <script>
           const array = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13];

           function isPrime(num) {
           if(num==0) return false;
           if(num<0) num=-1*num;//convert num to positive number
           for (let i = 2; i < num; i++) {
               if (num % i == 0) {
               return false;
               }
           }
           return true;
           }
           console.log(array.filter(isPrime)); // [2, 3, 5, 7, 11, 13]
       </script>
   </body>
</html>
```

Reduce

השיטה reduce() מבצעת פונקציית callback של "reducer" שסופק על ידי המשתמש בכל אלמנט של המערך, ומעבירה את ערך ההחזרה מהחישוב באלמנט הקודם. התוצאה הסופית של הפעלת המפחית על פני כל האלמנטים של המערך היא ערך בודד.

המפחית עובר דרך המערך אלמנט אחר אלמנט, בכל שלב מוסיף את ערך המערך הנוכחי לתוצאה מהשלב הקודם - עד שאין יותר אלמנטים להוסיף. <html>

```
<body>
    <script>
        const array1 = [1, 2, 3, 4];
        const reducer = (previousValue, currentValue) => previousValue + currentValue;

        // 1 + 2 + 3 + 4
        console.log(array1.reduce(reducer));
        // expected output: 10

        // 5 + 1 + 2 + 3 + 4
        console.log(array1.reduce(reducer, 5));
        // expected output: 15
    </script>
</body>
</html>
```

Reduce

אם initialValue לא מסופק, reduce method תפעל אחרת עבור מערכים בעלי אורך גדול מ-1, שווה ל-1 ו-0, כפי שמוצג בדוגמה הבאה:

```
<html>
   <body>
       <script>
           const getMax = (a, b) => Math.max(a, b);

           // callback is invoked for each element in the array starting at index 0
           [1, 100].reduce(getMax, 50); // 100
           [    50].reduce(getMax, 10); // 50

           // callback is invoked once for element at index 1
           [1, 100].reduce(getMax);     // 100

           // callback is not invoked
           [    50].reduce(getMax);     // 50
           [      ].reduce(getMax, 1);  // 1

           [      ].reduce(getMax);     // TypeError
       </script>
   </body>
</html>
```

Find

השיטה find() מחזירה את הערך של האלמנט הראשון במערך המסופק, המקיים את פונקציית הבדיקה שסופקה. אם אין ערכים העונים על פונקציית הבדיקה, Undefined מוחזר.

```
<html>
   <body>
       <script>
           const array1 = [5, 12, 8, 130, 44];
           const found = array1.find(element => element > 10);
           console.log(found);
           // expected output: 12
       </script>
   </body>
</html>
```

Find

The find method executes the callbackFn function once for each index of the array until the callbackFn returns a truthy value. If so, find immediately returns the value of that element. Otherwise, find returns undefined.

callbackFn is invoked for every index of the array, not just those with assigned values. This means it may be less efficient for sparse arrays, compared to methods that only visit assigned values.

If a thisArg parameter is provided to find, it will be used as the this value inside each invocation of the callbackFn. If it is not provided, then undefined is used.

Find

מצא אובייקט במערך לפי אחד המאפיינים שלו

```
<html>
   <body>
       <script>
           const inventory = [
               {name: 'apples', quantity: 2},
               {name: 'bananas', quantity: 0},
               {name: 'cherries', quantity: 5}
               ];

               function isCherries(fruit) {
               return fruit.name === 'cherries';
               }

               console.log(inventory.find(isCherries));
               // { name: 'cherries', quantity: 5 }
       </script>
   </body>
</html>
```

Find

Using arrow function and destructuring

```
<html>
    <body>
        <script>
            const inventory = [
            {name: 'apples', quantity: 2},
            {name: 'bananas', quantity: 0},
            {name: 'cherries', quantity: 5}
            ];

            const result = inventory.find( ({ name }) => name === 'cherries' );
            console.log(result) // { name: 'cherries', quantity: 5 }
        </script>
    </body>
</html>
```

findindex

שיטת findIndex() מחזירה את האינדקס של האלמנט הראשון במערך המקיים את פונקציית הבדיקה שסופקה. אחרת, הוא מחזיר -1, המציין שאף אלמנט לא עבר את המבחן.

```
<html>
   <body>
       <script>
           function isPrime(num) {
           for (let i = 2; num > i; i++) {
               if (num % i == 0) {
               return false;
               }
           }
           return num > 1;
           }
           console.log([4, 6, 8, 9, 12].findIndex(isPrime)); // -1, not found
           console.log([4, 6, 7, 9, 12].findIndex(isPrime)); // 2 (array[2] is 7)
       </script>
   </body>
</html>
```

findindex

The following example finds the index of a fruit using an arrow function:

```
<html>
   <body>
       <script>
           const fruits = ["apple", "banana", "cantaloupe", "blueberries", "grapefruit"];

           const index = fruits.findIndex(fruit => fruit === "blueberries");

           console.log(index); // 3
           console.log(fruits[index]); // blueberries
       </script>
   </body>
</html>
```

Some

שיטת some() בודקת אם לפחות אלמנט אחד במערך עובר את הבדיקה המיושמת על ידי הפונקציה שסופקה. הוא מחזיר true אם במערך הוא מוצא אלמנט שעבורו הפונקציה שסופקה מחזירה true; אחרת הוא מחזיר false. זה לא משנה את המערך.

```
<html>
   <body>
       <script>
           const array = [1, 2, 3, 4, 5];
           // checks whether an element is even
           const even = (element) => element % 2 === 0;

           console.log(array.some(even));
           // expected output: true
       </script>
   </body>
</html>
```

Some

הדוגמה הבאה בודקת אם אלמנט כלשהו במערך גדול מ-10.

```
<html>
   <body>
       <script>
           function isBiggerThan10(element, index, array) {
           return element > 10;
           }

           [2, 5, 8, 1, 4].some(isBiggerThan10);  // false
           [12, 5, 8, 1, 4].some(isBiggerThan10); // true
       </script>
   </body>
</html>
```

Some + array functions

בדיקת רכיבי מערך באמצעות פונקציות חצים

```
<html>
    <body>
        <script>
            [2, 5, 8, 1, 4].some(x => x > 10);  // false
            [12, 5, 8, 1, 4].some(x => x > 10); // true
        </script>
    </body>
</html>
```

Every

השיטה every() בודקת אם כל האלמנטים במערך עוברים את הבדיקה המיושמת על ידי הפונקציה שסופקה. זה מחזיר ערך בוליאני.

```
<html>
    <body>
        <script>
            const isBelowThreshold = (currentValue) => currentValue < 40;
            const array1 = [1, 30, 39, 29, 10, 13];
            console.log(array1.every(isBelowThreshold));
            // expected output: true
        </script>
    </body>
</html>
```

PART B

Numbers Dates and Timers JAVASCRIPT

2023 © Braintio ltd | Confidential

Numbers - good to know

```
<html>
   <body><script>
           console.log(90===90.0)//true
           console.log(Number('125'))//125
           console.log(+'125')//125
           console.log(Number.parseInt('50px'))//50
           console.log(Number.parseFloat('2.5'))//2
           console.log(Number.parseFloat('5.4'))//5.4
           console.log("========Number.isNaN37==========")
           console.log(Number.isNaN(20))//false
           console.log(Number.isNaN("23"))//false
           console.log(Number.isNaN(4/0))//false
           console.log("========isFinite==========")
           console.log(Number.isFinite(20))//true
           console.log(Number.isFinite("23"))//false
           console.log(Number.isFinite(4/0))//false
           console.log("=======isInteger==========")
           console.log(Number.isInteger(30))//true
           console.log(Number.isInteger(30.0))//true
           console.log(Number.isInteger(30/0))//false
       </script>
   </body>
</html>
```

Numbers - good to know

```
<html>
   <body>
       <h1>Math</h1>
       <script>
           console.log(Math.sqrt(49))// 7
           console.log(5**5)//25
           console.log(5**5)//25
           console.log(8**(1/3))//2
           console.log(Math.max(2,10))//10
           console.log(Math.min(2,10))//2
           console.log(Math.PI)//3.141592653589793
           console.log(Math.random())//0-1
           console.log(Math.trunc(Math.random()*10 + 1))//1-10
           let myRandom = (min, max)=>Math.trunc(Math.random()*(max-min) +1)+min
           console.log(myRandom(10,20))//10-20
       </script>
   </body>
</html>
```

Numbers - good to know

```
<html>
   <body>
       <h1>Math</h1>
       <script>
           console.log("===========rounding==========")
           console.log(Math.round(40.23))//40
           console.log(Math.ceil(23.2))//24
           console.log(Math.floor(23.7))//24 its for negetive too
           //rounding decimal
           console.log((2.7).toFixed(0))
           console.log((2.7).toFixed(3))//2.700
           console.log((1.768).toFixed(2))//1.77
           console.log((2.876).toFixed(2))//2.88
           //big numbers
           console.log(Number.MAX_SAFE_INTEGER)//9007199254740991
       </script>
   </body>
</html>
```

Dates

JavaScript Date Object lets us work with dates. By default, JavaScript will use the browser's time zone and display a date as a full text string.

`new Date()` creates a new date object with the current date and time. For example:

```
const d = new Date();
```

Dates

```
<html>
   <body>
       <h1>Date</h1>
       <script>
           let now = new Date();
           console.log(now)
           console.log(new Date("November 12, 2021"))
           console.log(new Date(2021,10,25,10,43,20))//year,month,day,hour,minuets, seconds
           //motnh : 0-11
           console.log(new Date(2021,10,25))//year,month,day
           console.log(now.getFullYear())
           console.log(now.getMonth())
           console.log(now.getDate())
           console.log(now.getDay())
           console.log(now.getHours())
           console.log(now.getMinutes())
           console.log(now.getSeconds())
           console.log(now.toISOString())
           console.log(now.getTime())// stamp
           //you can use setters
           now.setMonth(1)

       </script>
   </body>
</html>
```

Dates

```
<html>
   <body>
       <h1>Opertions With Dates </h1>
       <script>
          let day1= new Date(2021,10,25)
          let day2= new Date(2024,5,25)
          let calcNumOfdays = (d1, d2)=>Math.abs(d1-d2)/(1000*60*60*24)
          let days = calcNumOfdays(day1,day2)
          console.log(days)

       </script>
   </body>
</html>
```

setTimeout

The setTimeout() method calls a function after a number of milliseconds.

1 second = 1000 milliseconds.

Notes:

The setTimeout() is executed only once.

If you need repeated executions, use setInterval() instead.

Use the clearTimeout() method to prevent the function from starting.

setTimeout

```
<html>
   <body>
       <h1>set time out </h1>
       <script>
           let colors = function(){
               arr=["red","blue","green","black"]
               let random =Math.floor(Math.random()*4)//0-4
               console.log(random)
               document.body.style.background=arr[random]
           }
           setTimeout(colors,3000)//calling colors after 3000 secin
       </script>
   </body>
</html>
```

setTimeout

```
<html>
   <body>
       <h1>set time out </h1>
       <script>
           let colors = function(range){
               arr=["red","blue","green","black"]
               let random =Math.floor(Math.random()*range)//0-4
               console.log(random)
               document.body.style.background=arr[random]
           }
           setTimeout(colors,1000,4)//calling colors after 3000 seconds
       </script>
   </body>
</html>
```

setInterval

The `setInterval()` method calls a function at specified intervals (in milliseconds).

The `setInterval()` method continues calling the function until `clearInterval()` is called, or the window is closed.

1 second = 1000 milliseconds.

Example:

Display "Hello" every second (1000 milliseconds):

```
setInterval(function () {element.innerHTML += "Hello"}, 1000);
```

setInterval

```
<html>
   <body>
       <h1>set interval </h1>
       <script>
           let colors = function(range){
               arr=["red","blue","green","black"]
               let random =Math.floor(Math.random()*range)//0-4
               console.log(random)
               document.body.style.background=arr[random]
           }
           setInterval(colors,1000,4)//calling colors after every 1000 milliseconds = 1
second
       </script>
   </body>
</html>
```

setInterval

```
<html>
   <body>
       <h1>count timer </h1>
       <p id="timer">0</p>
       <script>
           function counting(){
               let time = document.getElementById("timer").innerText;
               time = Number(time)
               time++
               document.getElementById("timer").innerText = time
           }
           setInterval(counting,1000)
       </script>
   </body>
</html>
```

Timers practice

צור דף HTML שמציג בכל שנייה מספר בין 0-100

```
<html>
   <body>
       <script>





       </script>
   </body>
</html>
```

פתרון תרגול טיימרים

```
<html>
   <body>
       <script>
           let range = 100;
           let nums = function(){
               let random =Math.floor(Math.random()*range)//0-100
               document.body.innerText=random;
           }
           setInterval(nums,1000)//calling nums function every 1
seconds
       </script>
   </body>
</html>
```

PART B

DOM PART B

JAVASCRIPT

2023 © Braintio ltd | Confidential

דרכים נוספות to attached events

```
<html>
   <body>
       <h1>More ways to attached events</h1>
       <p id="e1">My name is e1</h1>
       <p id="e2">My name is e2</h1>
       <script>
       let e1 = document.getElementById("e1")
       let e2 = document.getElementById("e2")

       e1.addEventListener('mouseenter', function(e){
           alert("Hi, mouse enter to e1")
       })
       e2.onmouseenter = function(e){
           alert("Hi, mouse enter to e2")
       }
       </script>
   </body>
</html>
```

הסרת event

```
<html>
   <body>
       <h1>More ways to attached events</h1>
       <p id="e1">My name is e1</h1>
       <br><br>
       <p id="e2">My name is e2</h1>

       <script>

       let e1 = document.getElementById("e1")
       let e2 = document.getElementById("e2")

       function alertE1() {
           alert("Hi, mouse enter to e1 and remove mouseenter event Listener")
           e1.removeEventListener('mouseenter',alertE1)
       }
       e1.addEventListener('mouseenter', alertE1)

       e2.onmouseenter = function(e){
           alert("Hi, mouse enter to e2")
       }
       </script>
   </body>
</html>
```

setAttribute

מגדיר את הערך של תכונה באלמנט שצוין. אם התכונה כבר קיימת, הערך מתעדכן; אחרת מתווספת תכונה חדשה עם השם והערך שצוינו.

```
Element.setAttribute(name, value);
```

name

DOMString המציין את שם התכונה שיש להגדיר את הערך שלה. שם התכונה מומר אוטומטית לאותיות קטנות כאשר setAttribute() נקרא ברכיב HTML במסמך HTML.

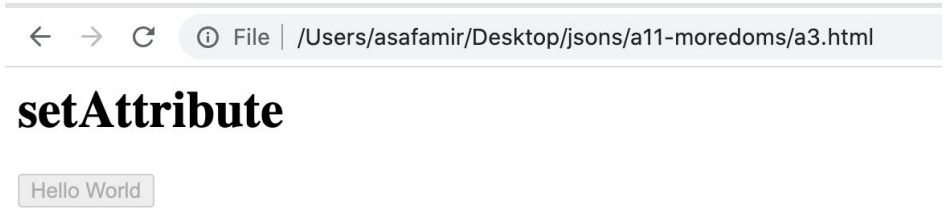
value

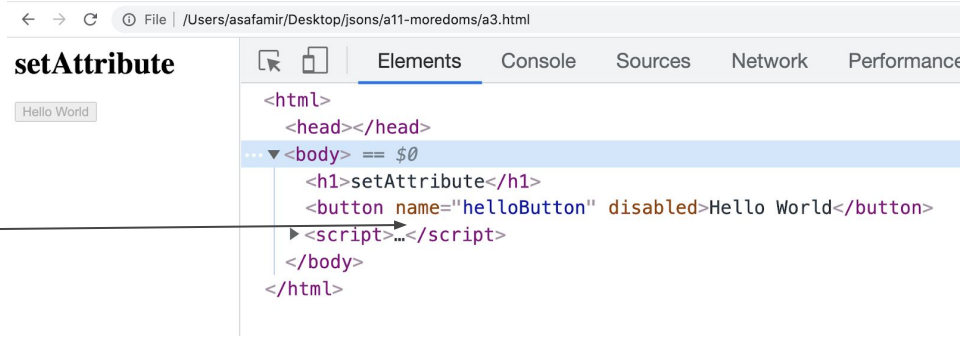
DOMString המכיל את הערך להקצאה לתכונה. כל ערך שאינו מחרוזת שצוין מומר אוטומטית למחרוזת.

setAttribute

```html
<html>
    <body>
        <h1>setAttribute</h1>
        <button>Hello World</button>

        <script>
        //Element.setAttribute(name, value);
        var b = document.querySelector("button");
        b.setAttribute("name", "helloButton");
        b.setAttribute("disabled", "");
        </script>
    </body>
</html>
```

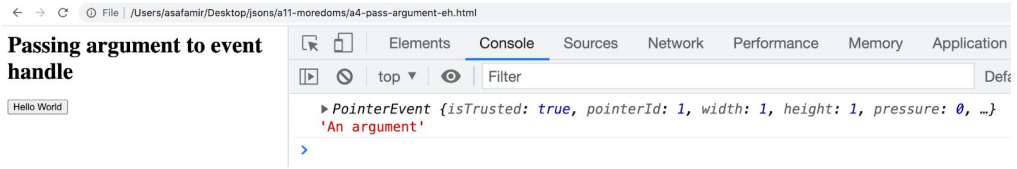




setAttribute

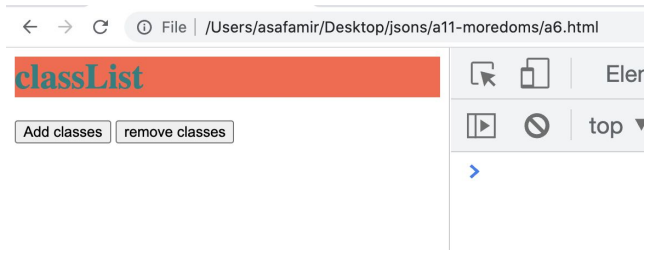
```
<html>
   <body>
       <h1>Passing argument to event handle</h1>
       <button>Hello World</button>

       <script>
           // create a function
           function event_handler(event, arg) {
           console.log(event, arg);
           }
           // Assign the listener callback to a variable
           var makeClick = (event) => event_handler(event, 'An argument');
           el = document.querySelector('button');
           el.addEventListener('click', makeClick);
       </script>
   </body>
</html>
```



הוספה והסרה של קלאסים

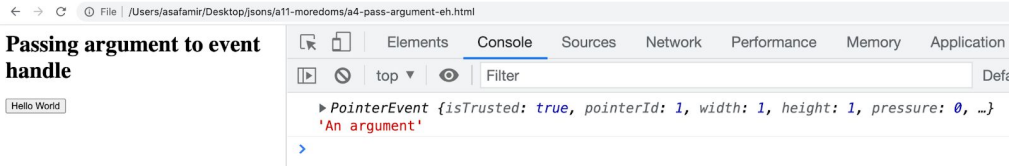
```html
<html>
   <head><style>
        .red{
            color: teal;
        }
        .teal{
            background: tomato;
        }
   </style></head>
   <body>
        <h1>classList</h1>
        <button onclick="addClasses()">Add classes</button>
        <button onclick="removeClasses()">remove classes</button>
        <script>
            function addClasses(){
                let h1 = document.querySelector("h1")
                h1.classList.add('red', 'teal');
            }
            function removeClasses(){
                let h1 = document.querySelector("h1")
                h1.classList.remove('red', 'teal');
            }
            // logo.classList.add('a','b','c')
            // logo.classList.remove('a','b','c')
            // logo.classList.toggle()
            // logo.classList.contains()
        </script>
   </body>
</html>
```



העברת פרמטרים ל event handler

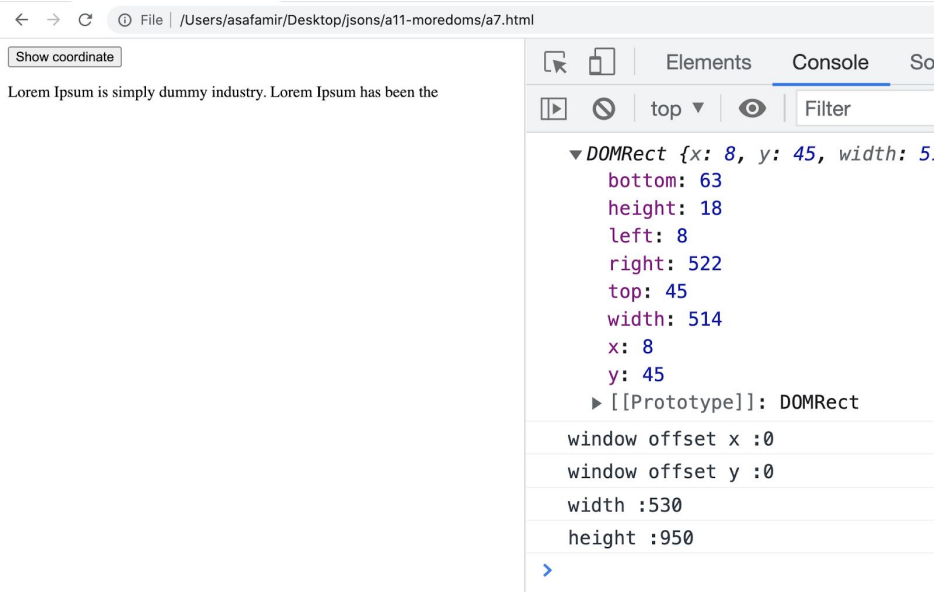
```html
<html>
    <body>
        <h1>Passing argument to event handle</h1>
        <button>Hello World</button>

        <script>
            // create a function
            function event_handler(event, arg) {
            console.log(event, arg);
            }
            // Assign the listener callback to a variable
            var makeClick = (event) => event_handler(event, 'An argument');
            el = document.querySelector('button');
            el.addEventListener('click', makeClick);
        </script>
    </body>
</html>
```



Windows coordinate - good to know

```html
<html>
    <body>
        <div>
            <button onclick="show()">Show coordinate</button>
        </div>
        <div id="box1" class="box">
            <p>
                Lorem Ipsum is simply dummy
                industry. Lorem Ipsum has been the
            </p>
        </div>
        <script>
            function show(){
                let box=document.getElementById("box1")
                let coords = box.getBoundingClientRect();
                console.log(coords)
                console.log('window offset x :' + window.pageXOffset)
                console.log('window offset y :' + window.pageYOffset)
                console.log('width :' +
document.documentElement.clientWidth)
                console.log('height :' +
document.documentElement.clientHeight)
            }
        </script>
    </body>
</html>
```



smooth gallery הצגת דמו

```
box.scrollIntoView({behavior: 'smooth'})
```

Show and explain on demo

PART B

OOP

JAVASCRIPT

2023 © Braintio ltd | Confidential

Old way to defining an object

```
<html>
   <body>
       <script>
           // This works well enough, but it is a bit long-winded;
           // if we know we want to create an object,
           // why do we need to explicitly
           // create a new empty object and return it?

           function createNewPerson(name) {
           const obj = {};
           obj.name = name;
           obj.greeting = function() {
               alert('Hi! I\'m ' + obj.name + '.');
           };
           return obj;
           }

           const salva = createNewPerson('Salva');
           salva.name;
           salva.greeting();
       </script>
   </body>
</html>
```

The constructor function

```
<html>
   <body>
       <script>
//Replace your previous function with the following
           function Person(name) {
           this.name = name;
           this.greeting = function() {
           alert('Hi! I\'m ' + this.name + '.');
       };
   }
   let person1 = new Person('Bob');
   let person2 = new Person('Sarah');
   person1.name
   person1.greeting()
   person2.name
   person2.greeting()
       </script>
   </body>
</html>
```

The constructor function is JavaScript's version of a class. Notice that it has all the features you'd expect in a function, although it doesn't return anything or explicitly create an object — it basically just defines properties and methods.
Notice also the this keyword being used here as well — it is basically saying that whenever one of these object instances is created, the object's name property will be equal to the name value passed to the constructor call, and the greeting() method will use the name value passed to the constructor call too.

```
After the new objects have been created,
the person1 and person2 variables contain the following
objects:

    {
        name: 'Bob',
        greeting: function() {
        alert('Hi! I\'m ' + this.name + '.');
    }
   }

   {
        name: 'Sarah',
        greeting: function() {
        alert('Hi! I\'m ' + this.name + '.');
       }
    }
```

81

פונקציית הבנאי (קונסטרוקטור)

```
<html>
   <body>
   <script>
       function Person(first, last, age, gender, interests) {
       this.name = {
           first : first,
           last : last
       };
       this.age = age;
       this.gender = gender;
       this.interests = interests;
       this.bio = function() {
           alert(this.name.first + ' ' + this.name.last + ' is ' + this.age +
' years old. He likes ' + this.interests[0] + ' and ' + this.interests[1] +
'.');
       };
       this.greeting = function() {
           alert('Hi! I\'m ' + this.name.first + '.');
       };
       }
      let person1 = new Person('Bob', 'Smith', 32, 'male', ['music', 'skiing']);
       person1['age']
       person1.interests[1]
       person1.bio()
   </script>

   </body>
</html>
```

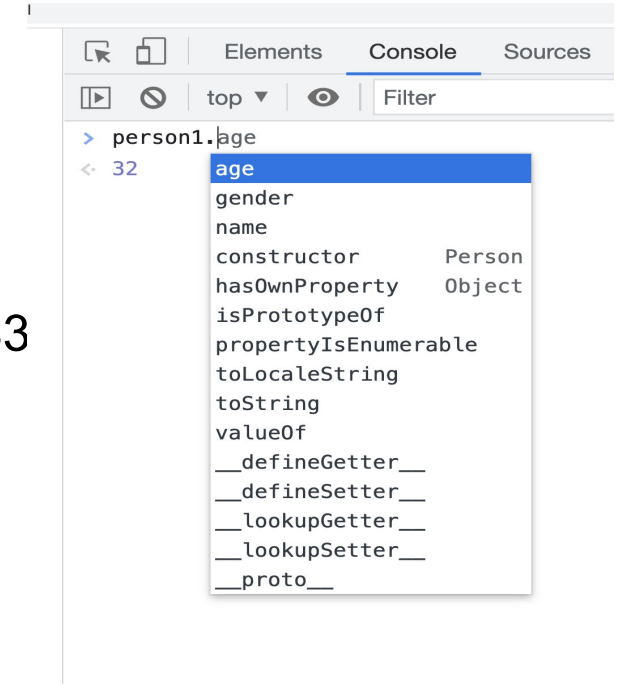
כעת אתה יכול לראות שאתה יכול לגשת למאפיינים ולשיטות
בדיוק כמו שעשינו בעבר - נסה את אלה במסוף ה-JS שלך:

Prototypes mechanism

Prototypes הם המנגנון שבאמצעותו אובייקט
JavaScript יורשים תכונות אחד מהשני.
נסביר כיצד שרשרות Prototypes
עובדות ונראה כיצד Prototype
יכול לשמש להוספת שיטות לבנאים קיימים.
בדוגמה זו, הגדרנו פונקציית בנאי.

```
<html>
   <body>
       <script>
   function Person(first, last, age, gender, interests) {
       // property and method definitions
       this.name = {
       'first': first,
       'last' : last
       };
       this.age = age;
       this.gender = gender;
   }
let person1 = new Person('Bob', 'Smith', 32, 'male', ['music',
'skiing']);
//If you type "person1." into your JavaScript console, you
should see the browser try to auto-complete this with the
member names available on this object:

       </script>
   </body>
</html>
```



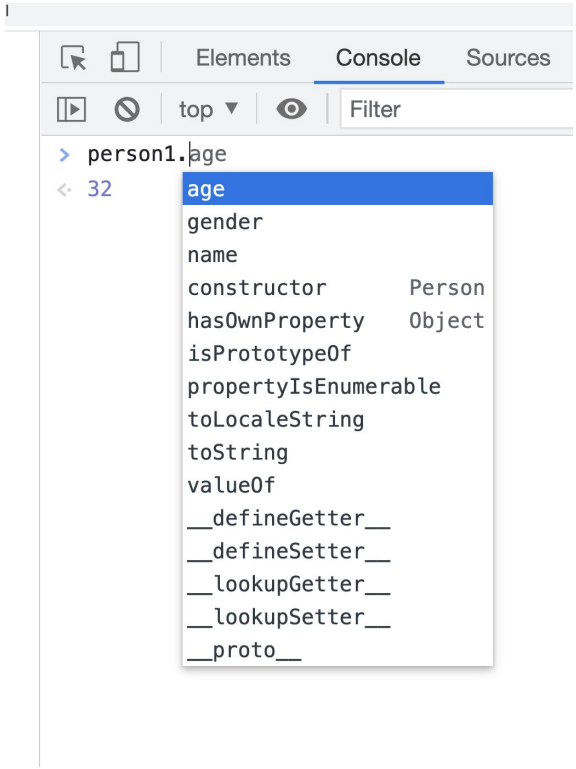
83

Prototypes mechanism

```
<html>
    <body>
        <script>
    function Person(first, last, age, gender, interests) {
        // property and method definitions
        this.name = {
        'first': first,
        'last' : last
        };
        this.age = age;
        this.gender = gender;
    }
let person1 = new Person('Bob', 'Smith', 32, 'male', ['music',
'skiing']);
//If you type "person1." into your JavaScript console, you
should see the browser try to auto-complete this with the
member names available on this object:

        </script>
    </body>
</html>
```

ברשימה זו, תראה את החברים המוגדרים בקונסטרוקטור של Person — person1() — שם, גיל, מין, תחומי עניין, ביוגרפיה וברכה.
עם זאת, תראה גם - toString, valueOf וכו' - אלה מוגדרים באובייקט אב הטיפוס של אובייקט אב הטיפוס של person1, שהוא Object.prototype.



Prototypes mechanism

מה קורה אם קוראים למתודה ב-person1, שבעצם מוגדרת ב-Object.prototype? לדוגמה:

person1.valueOf()

valueOf() returns the value of the object it is called on. In this case, what happens is:

- The browser initially checks to see if the person1 object has a valueOf() method available on it, as defined on its constructor, Person(), and it doesn't.
- So the browser checks to see if the person1's prototype object has a valueOf() method available on it. It doesn't, then the browser checks person1's prototype object's prototype object, and it has. So the method is called, and all is good!

The prototype property good to know

look at the Object reference page,
In the left hand side there is a large number of properties and methods — many more than the number of inherited members. Some are inherited, and some aren't — why is this?

The inherited ones are the ones defined on the prototype property (you could call it a sub-namespace) — that is, the ones that begin with Object.prototype., and not the ones that begin with just Object.

The prototype property's value is an object.
Object.prototype.toString(), Object.prototype.valueOf(), etc., are available to any object types that inherit from Object.prototype, including new object instances created from the Person() constructor.

Object.is(), Object.keys(), and other members not defined inside the prototype bucket, are not inherited by object instances or object types that inherit from Object.prototype. They are methods/properties available just on the Object() constructor itself.

הורשה ב JavaScript

עד כה ראינו ירושה מסוימת בפעולה - ראינו איך שרשראות אב טיפוס עובדות, ואיך חברים עוברים בירושה במעלה שרשרת. אבל בעיקר זה כלל פונקציות מובנות בדפדפן. איך יוצרים אובייקט ב-JavaScript שיורש מאובייקט אחר?

```
<html>
   <body>
      <script>
            function Person(first, last, age, gender, interests) {
            this.name = {
                first,
                last
            };
            this.age = age;
            this.gender = gender;
            this.interests = interests;
            };
            //The methods are all defined on
            //the constructor's prototype. For example:
            Person.prototype.greeting = function() {
            alert('Hi! I\'m ' + this.name.first + '.');
            };
             function Teacher(first, last, age, gender,
             interests, subject) {
             Person.call(this, first, last, age, gender, interests);
             this.subject = subject;
             }
      </script>
   </body>
</html>
```

הורשה ב JavaScript

the call() function.

פונקציה זו מאפשרת לך לקרוא לפונקציה שהוגדרה במקום אחר, אך בהקשר הנוכחי. הפרמטר הראשון מציין את הערך של זה שבו אתה רוצה להשתמש בעת הפעלת הפונקציה, והפרמטרים האחרים הם אלה שיש להעביר לפונקציה כאשר היא מופעלת.

אנחנו רוצים שהקונסטרוקטור של Teacher() ייקח את אותם פרמטרים כמו הבנאי Person() שממנו הוא יורש, אז אנחנו מציינים את כולם כפרמטרים בהפעלת call().

השורה האחרונה בתוך הקונסטרוקטור מגדירה את מאפיין הנושא החדש שיהיו למורים, אשר לאנשים גנרי אין.

ירושה מבנאי ללא פרמטרים

שים לב שציינו רק את זה בתוך call() - לא אחר.
פרמטרים נדרשים מכיוון שאנו לא יורשים תכונות כלשהן מהאב המוגדרות באמצעות פרמטרים.

שים לב שאם הבנאי שאתה יורש ממנו לא לוקח את ערכי מאפיינים שלו מפרמטרים, אין צורך לציין אותם כארגומנטים נוספים ב-call().
אז, למשל, אם היה לך משהו ממש פשוט כמו זה:

```
function Brick() {
this.width = 10;
this.height = 20;
}
//You could inherit the width and height properties
//by doing this (as well as the other steps described below,
// of course):
function BlueGlassBrick() {
 Brick.call(this);
 this.opacity = 0.5;
 this.color = 'blue';
}
```

EcmaScript 2015 classes

מחלקות ECMAScript 2015 ECMAScript 2015 מציג תחביר מחלקות ל-JavaScript כדרך לכתוב מחלקות לשימוש חוזר באמצעות תחביר נקי יותר, הדומה יותר למחלקות ב-C++ או ב-Java.
בחלק זה נמיר את הדוגמאות של האדם והמורה מהירושה של אב טיפוס לשיעורים, כדי להראות לך איך זה נעשה.

```
//Let's look at a rewritten version of the Person example, class-style:
class Person {
constructor(first, last, age, gender, interests) {
    this.name = {
    first,
    last
    };
    this.age = age;
    this.gender = gender;
    this.interests = interests;
}

greeting() {
    console.log(`Hi! I'm ${this.name.first}`);
};

farewell() {
    console.log(`${this.name.first} has left the building. Bye for now!`);
};
}
```

EcmaScript 2015 classes

הצהרת ה class מציין שאנו יוצרים כיתה חדשה.
constructor() מחלקת ה-Person שלנו.
greeting() ו-farewell() הן פונקציות ה class. כל המתודות שאתה רוצה לשייך למחלקה מוגדרות בתוכה, אחרי הבנאי. בדוגמה זו, השתמשנו בתבנית מילולית במקום בשרשור מחרוזות כדי להקל על הקריאה של הקוד.
כעת אנו יכולים ליצור מופעי אובייקט בדיוק באותו אופן שעשינו קודם:

```
let han = new Person('Han', 'Solo', 25, 'male', ['Smuggling']);
han.greeting();
// Hi! I'm Han

let leia = new Person('Leia', 'Organa', 19, 'female', ['Government']);
leia.farewell();
// Leia has left the building. Bye for now
```

EcmaScript 2015 classes inheritance good to know

בחלק זה ניצור class Teacher שמרחיב את class Person
כדי ליצור הרחבה של מחלקה אנו משתמשים במילת המפתח extends

```
class Teacher extends Person {
constructor(subject, grade) {
    super(); // Now 'this' is initialized by calling the parent constructor.
    this.subject = subject;
    this.grade = grade;
    }
}
```

92

EcmaScript 2015 classes inheritance

```
class Person{
    constructor(first, last, age, gender, interests) {
    this.name = {
        first,
        last
    };
    this.age = age;
    this.gender = gender;
    this.interests = interests;
    }
}
```

EcmaScript 2015 classes inheritance

מכיוון שהאופרטור super() הוא למעשה בנאי מחלקת האב, העברת ארגומנטים הנחוצים של בנאי מחלקת האב יאתחל גם את מאפייני מחלקת האב בתת המחלקה שלנו, ובכך יורש אותה:

```
class Teacher extends Person {
constructor(first, last, age, gender, interests, subject, grade) {
        super(first, last, age, gender, interests);

    // subject and grade are specific to Teacher
    this.subject = subject;
    this.grade = grade;
}
}
let snape = new Teacher('Severus', 'Snape', 58, 'male', ['Potions'], 'Dark arts', 5);
snape.greeting(); // Hi! I'm Severus.
snape.farewell(); // Severus has left the building. Bye for now.
snape.age; // 58
snape.subject; // Dark arts
```

EcmaScript 2015 classes getters and setters

ייתכנו מקרים שבהם נרצה לשנות את ערכי המאפיינים במחלקות שאנו יוצרים,
או פעמים בהן איננו יודעים מה יהיה הערך הסופי של תכונה מסוימת.
אם נשתמש בדוגמה של המורה, ייתכן שלא נדע איזה נושא המורה ילמד לפני שניצור אותם, או הנושא שלהם עשוי להשתנות.

בואו נשדרג את class המורה עם getters and setters.

getter מחזיר את הערך הנוכחי של המשתנה, setter המתאים לו משנה את ערך המשתנה לזה שהוא מגדיר.

EcmaScript 2015 classes getters and setters

```
class Teacher extends Person {
constructor(first, last, age, gender, interests, subject, grade) {
    super(first, last, age, gender, interests);
    // subject and grade are specific to Teacher
    this._subject = subject;
    this.grade = grade;
}
get subject() {
    return this._subject;
}
set subject(newSubject) {
    this._subject = newSubject;
}
}
```

EcmaScript 2015 classes getters and setters

כדי להציג את הערך הנוכחי של המאפיין _subject של אובייקט ה-snap אנחנו יכולים להשתמש בשיטת snape.subject getter.

כדי להקצות ערך חדש למאפיין _subject נוכל להשתמש ב-שיטת הגדרת snape.subject="ערך חדש".

הדוגמה שלהלן מציגה את שתי התכונות בפעולה:

בדוק את ערך ברירת המחדל

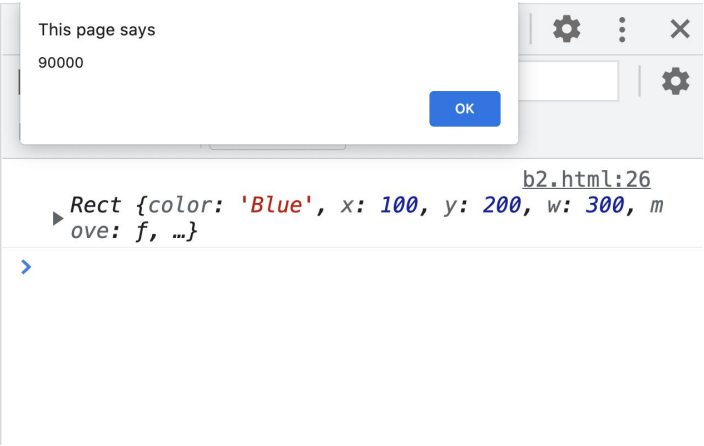
```
console.log(snape.subject) // Returns "Dark arts"

// Change the value
snape.subject = "Balloon animals" // Sets _subject to "Balloon animals"

// Check it again and see if it matches the new value
console.log(snape.subject) // Returns "Balloon animals"
```

Do it yourself 48 (Home practice booklet) with ecmascript 2015

1. צור class RectAngle שיורש מ shape. הוסף שתי תכונות ל class. רוחב המלבן w, וגובה המלבן h.
2. הוסף למחלקה פונקציה שנקראת area מחזירה את שטח המלבן (w * h)
3. צור מופע מחלקה עם צבע אדום, x 100, y 200, רוחב 300 וגובה 300
4. הפעל את פונקציית השטח והדפס את המחלקה ל console



PART B

Asynchronous JAVASCRIPT

2023 © Braintio ltd | Confidential

Asynchronous Javascript

המטרה של javascript אסינכרוני להתמודד עם משימות ריצה ארוכות שרצות ברקע.

דוגמה: אחזור נתונים מהשרת - Ajax

סינכרוני הוא קוד הפועל שורה אחר שורה בסדר המדויק.

```
let x = 6;
let y = 4;
let sum = x+y;
console.log(sum);
```

Asynchronous Javascript

```
<html>
    <body>
        <p id="details"></p>
        <button onclick="details()">click</button>
        <script>
            function details() {
            let p = document.getElementById("details")
            //asynchronous
            //the main code is not blocked
            setTimeout(function() {
                p.innerText="Asaf"
            }, 3000);
            p.style.background="blue"
        }
        </script>
    </body>
</html>
```

Asynchronous Javascript

```
<html>
    <body>
        <img src="" class="cat">
        <button onclick="details()">click</button>
        <script>
            function details() {
            let img = document.querySelector(".cat")
            //img.src is asynchronous
            img.src="cat.png"
            img.addEventListener('load', function(){
                img.classList.add("someclass")
            })
        }
        </script>
    </body>
</html>
```

עם Ajax אנחנו יכולים:

1.) קרא נתונים מהשרת

2.) עדכן דף אינטרנט מבלי לטעון מחדש

3.) שלח נתונים לשרת ברקע

API

API : ממשק תכנות יישומים: חתיכת תוכנה. זה יכול

להיות בשימוש על ידי תוכנה אחרת, על מנת לאפשר יישום

לדבר אחד עם השני

web api :זו אפליקציה שפועלת על שרת, מקבלת בקשות לנתונים ושולחת נתונים בחזרה כתגובה

There are lots of api's - weather data, flight ,currency, google play ,google maps and more

Api data format

Ajax - ה-x ב-ajax מייצג xml.אבל היום אף API לא משתמש בנתוני xml יותר.

רוב ה-API כיום משתמש ב-JSON API. json הוא פורמט הנתונים הפופולרי ביותר כיום, מכיוון שהוא אובייקט js שממיר

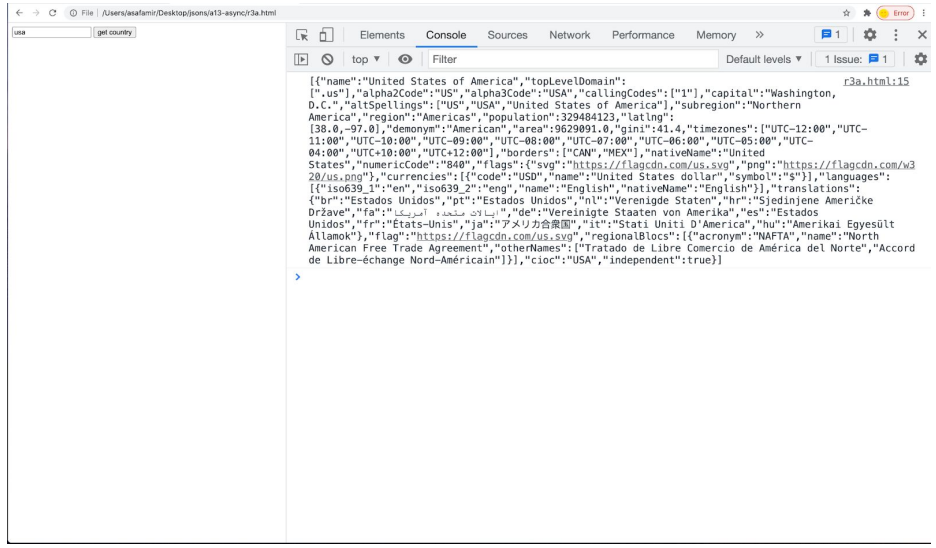
למחרוזת.

לכן קל מאוד לשלוח דרך האינטרנט ולהשתמש ב-javascript ברגע שמגיעים נתונים.

דוגמה 1 דרך ישנה Ajax

```
<html>
   <body>
       <input type="text" id="country">
       <button onclick="getCountryOnclick()">get country</button>
       <img src="" id="flag">
       <script>
           //old way.still exist
           let getCountry = function details(country) {
               let req =  new XMLHttpRequest();
               //do it in background
               req.open('GET', 'https://restcountries.com/v2/name/' + country)
               req.send()
               req.addEventListener('load',function(){
                    console.log(this.responseText)

       </script>
   </body>
</html>
```



דוגמה 2 דרך ישנה Ajax

```
<html>
    <body>
        <input type="text" id="country">
        <button onclick="getCountryOnclick()">get country</button>
        <p id="country"></p>
        <p id="region"></p>
        <p id="population"></p>
        <p id="language"></p>
        <p id="currency"></p>
        <img src="" id="flag">
```

דוגמה 2 דרך ישנה Ajax

```
<script>
    //old way.still exist
    let getCountry = function details(country) {
       let req =  new XMLHttpRequest();
       //do it in background
       req.open('GET', 'https://restcountries.com/v2/name/' + country)
       req.send()
       req.addEventListener('load',function(){
            //console.log(this.responseText)
            let [data] = JSON.parse(this.responseText)
            console.log(data)
            let img = document.getElementById("flag")
            let country = document.getElementById("country")
            let region =  document.getElementById("region")
            let population =  document.getElementById("population")
            let currency =  document.getElementById("currency")
            let language =  document.getElementById("language")
```

דוגמה 2 דרך ישנה Ajax

```
                    img.src = data.flag;
                    country.innerText = "country:" + data.name
                    region.innerText = "region:" + data.region
                    population.innerText = "population:" + data.population
                    currency.innerText = "currency:" + data.currencies[0].name
                    language.innerText = "language:" + data.languages[0].name
                });
        }

        function getCountryOnclick(){
            let country = document.getElementById("country").value
            getCountry(country)
        }
        </script>
    </body>
</html>
```

Promise

מהו promise?

promise הוא אובייקט JavaScript המקשר בין יצירת קוד לצורך קוד.

זה עלול לקחת זמן צריך לחכות לתוצאה.

אובייקט **promise** מייצג את ההשלמה (או הכישלון) בסופו של דבר של פעולה אסינכרונית ואת הערך הנובע ממנה.

מדוע ומתי אנו משתמשים בו?

promises הן הבחירה האידיאלית לטיפול בפעולות אסינכרוניות בצורה הפשוטה ביותר. הם יכולים להתמודד עם מספר פעולות אסינכרוניות בקלות ולספק טיפול טוב יותר בשגיאות מאשר התקשרויות חוזרות ואירועים.

תארו לעצמכם פונקציה, createAudioFileAsync(), אשר יוצרת באופן אסינכרוני קובץ קול בהינתן רשומת תצורה ושתי פונקציות callback, האחת נקראת אם קובץ האודיו נוצר בהצלחה, והשנייה נקראת אם מתרחשת שגיאה.

2023 © Braintio ltd | Confidential

דוגמא Promise

```
function successCallback(result) {
  console.log("Audio file ready at URL: " + result);
}

function failureCallback(error) {
  console.error("Error generating audio file: " + error);
}
function audioSettings () {
   //...
}

createAudioFileAsync(audioSettings, successCallback, failureCallback);
```

or

```
createAudioFileAsync(audioSettings).then(successCallback, failureCallback);
```


2023 © Braintio ltd | Confidential

Fetch Api

ה-Fetch API מספק ממשק לאחזור משאבים (כולל ברחבי הרשת). זה ייראה מוכר לכל מי שהשתמש ב-XMLHttpRequest, אבל ה-API החדש מספק מערך תכונות חזק וגמיש יותר.

המתודה fetch() לוקחת ארגומנט חובה אחד, הנתיב למשאב שברצונך לאחזר. היא מחזירה promise שמתייחס לתגובה לבקשה זו

אובייקט הpromise מייצג את ההשלמה (או הכישלון) בסופו של דבר של פעולה אסינכרונית ואת הערך הנובע ממנה.

Promise example-demo 1

Fetch return promise

```
let getCountry = function (name){
    //phase 1 :pending
    //the promise is settled
    //its fullfilled or rejected state
    //handle fullfilled promise
    let req = fetch(`https://restcountries.com/v2/name/${name}`)
    .then(function(response){
        console.log(response)
        //the data is in the body
        //need to call the json
        return response.json()//available in all result value.
        //The response.json() its asynchronous function. so its return promise

    }).then(function(data){
        console.log(data)
    })
    }
getCountry('usa')
</script>
```

Promise example-show demo 2

Promise arrow function-explain demo 3

```
<script>
let renderCountry = function (data) {
    let html =`
            <div>
                <img width="200" height="100" src="${data.flag}" />
                <p id="country">${data.flag}</p>
                <p id="region">${data.region}</p>
                <p id="population">${data.population}</p>
                <p id="language">${data.languages[0].name}</p>
                <p id="currency">${data.currencies[0].name}</p>
            </div>
    let div = document.querySelector(".flags")
    div.insertAdjacentHTML('beforeend', html)
    div.style.opacity = 1
}
let getCountry = function (name){
    let req = fetch(`https://restcountries.com/v2/name/${name}`)
    .then(response=>response.json())
    .then(data =>renderCountry(data[0]));
}
getCountry('usa')
</script>
```

Chaining promises

Promise example-show demo 4

errors promises

Promise example-show demo 5

errors promises

Promise example-show demo 6

finally promises

Promise example-show demo 7

More errors

Promise example-show demo 8

Create Promises

```
var promise = new Promise(function(resolve, reject){
// do thing (possibly asy) - smile, cry, .....bla
    if( /* every thing is ok - she marry u  */true ){
        resolve("Every thing work");
    }
    else if(/* she doesnt want u     */false){
        reject(Error("she broke my heart"));
    }
    });
    var a = promise.then(function(result){
    console.log(result);
    }, function(err){
    console.log(err);
});
```

Create promise

Promise example-show demo 9

Create wait promise

Promise example-show demo 10

Promises good to know

```
var promise = new Promise(function(resolve, reject){
resolve(1);
});

promise.then(function(val){
console.log(val);
return val + 2;

}).then(function(val){
console.log(val);
return val + 5;
}).then(function(val){
console.log(val);// ????
});
```

Promises good to know

```
var promise = new Promise(function(resolve, reject){
resolve(1);
});

promise.then(function(val){
console.log(val);
return val + 2;

}).then(function(val){
console.log(val);
return val + 5;
}).then(function(val){
console.log(val);// ????
});

var promise = new Promise(function(resolve, reject){
resolve(1);
});

promise.then(function(val){
console.log(val);
return val + 2;
}).then(function(val){
console.log(val);
return val + 5;
}).then(function(val){
console.log(val);// 8
});
```

Promises good to know

```
<html>
<head></head>
<body>hello</body>
<script>
function get(url) {
// Return a new promise.
return new  Promise(function(resolve, reject) {
// Do the usual XHR stuff
var req = new XMLHttpRequest();
req.open('GET', url, true);
req.onload = function() {
// This is called even on 404 etc
// so check the status
if (req.DONE && req.status == 200) {
// Resolve the promise with the response text
resolve(req.response);
}
else {// Otherwise reject with the status text, which will hopefully be a meaningful error
        reject(Error(req.statusText));
}
};
// Handle network errors
req.onerror = function() {
reject(Error("Network Error"));
};
// Make the request
req.send();
});
}
get('http://ilemon.mobi/site1/b.php')
.then(function(response){
console.log(response);
});
</script>
</html>
```

Promises good to know

```
<html>
<body>Promises</body>
<script>
function get(url) {
// Return a new promise.
return new  Promise(function(resolve, reject) {
// Do the usual XHR stuff
var req = new XMLHttpRequest();
req.open('GET', url, true);
req.onload = function() {
// This is called even on 404 etc
// so check the status
if (req.DONE && req.status == 200) {
// Resolve the promise with the response text
resolve(req.response);
}
else {
// Otherwise reject with the status text
// which will hopefully be a meaningful error
reject(Error(req.statusText));
}
};
// Handle network errors
req.onerror = function() {
reject(Error("Network Error"));
};
// Make the request
req.send();
});
}
get('http://ilemon.mobi/site1/b.php')
.then(function(response){
console.log(response);
});
</script>
</html>
```

Promises good to know

```
function get(url) {
return new  Promise(function(resolve, reject) {
var req = new XMLHttpRequest();
req.open('GET', url, true);
req.onload = function() {
if (req.status == 200) {
resolve(req.response);
}
else {
reject(Error(req.statusText));
}
};
req.onerror = function() {
reject(Error("Network Error"));
};
// Make the request
req.send();
});
}
```

Promises good to know

```
function getJson(url){
return get(url).then(JSON.parse);
}
//get json still return promise
 getJson('http://ilemon.mobi/site1/b.php')
.then(function(response){
console.log(response);
});
```

Promises good to know

```
<html>
<body>Promises</body>
<script type="text/javascript">
function get(url) {
return new  Promise(function(resolve, reject)
{
// Do the usual XHR stuff
var req = new XMLHttpRequest();
req.open('GET', url, true);
req.onload = function() {
if (req.status == 200) {
// Resolve the promise with the response text
}
else {
reject(Error(req.statusText));
}
};
req.onerror = function() {
reject(Error("Network Error"));
};
// Make the request
req.send();
});
}
function getJson(url){
return get(url).then(JSON.parse);
}
var url =
'http://api.open-notify.org/astros.json';
getJson(url).then(function(response){
console.log(response);
});
</script>
</html>
```

PART B

AWAIT JAVASCRIPT

2023 © Braintio ltd | Confidential

Async + await

Async ו await מקלים על כתיבה של Promises:

async גורם לפונקציה להחזיר Promise

await גורם לפונקציה לחכות Promise

בקר בדוגמה:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_async2

2023 © Braintio ltd | Confidential

131

Await

await operator משמש להמתנה ל Promise.

ניתן להשתמש בו רק בתוך פונקציית async בתוך קוד JavaScript רגיל; עם זאת ניתן להשתמש בו בפני עצמו עם מודולי JavaScript.

הביטוי await גורם לביצוע פונקציית async להשהות עד להסדרה של Promise (כלומר, מימוש או דחיה), ולחידוש הביצוע של פונקציית async לאחר מילוי. כאשר מתחדשים, הערך של ביטוי await הוא זה של promise שהצליחה.

אם ההבטחה נדחית, ביטוי async זורק את הערך שנדחה.

אם הערך של הביטוי שאחרי האופרטור await אינו Promise, הוא מומר ל Promise שנפתרה והצליחה.

ה await מפצלת את זרימת הביצוע, ומאפשרת למתקשר של פונקציית האסינכרון לחדש את הביצוע. לאחר שה- await דוחה את המשך פונקציית ה async, מתבצעת ביצוע של הצהרות עוקבות. אם המתנה זו היא הביטוי האחרון שבוצע על ידי הפונקציה שלו, הביצוע ממשיך על ידי החזרת לקורא של הפונקציה ה promise ממתינה להשלמת הפונקציה של המתנה וחידוש הביצוע של אותו מתקשר.

2023 © Braintio ltd | Confidential

Await

Awaiting a promise to be fulfilled.
If a Promise is passed to an await expression, it waits for the Promise to be fulfilled and returns the fulfilled value.

```
<html>
<body><h1>Hello await</h1></body>
<script>
function resolveAfter2Seconds(x) {
   return new Promise(resolve => {
     setTimeout(() => {
       resolve(x);
     }, 2000);
   });
 }
  async function f1() {
   var x = await resolveAfter2Seconds(10);
   console.log(x); // 10
 }
  f1();
 </script>
</html>
```


2023 © Braintio ltd | Confidential

133

Await

```
<html>
<body><h1>Await</h1></body>
<script>
   const delay = seconds => {
   return new Promise(
   resolve => setTimeout(resolve, seconds*1000)
   )};
   const countToFive = async() => {
   console.log("one");
   await delay(1);
   console.log("Two");
   await delay(1);
   console.log("three");
   await delay(1);
   };
countToFive();
</script>
</html>
```

134